Problem 1

$$\begin{aligned}\mathbb{E}[x] &= \int_{-\infty}^{\infty} x \cdot \sum_k \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)\,\mathrm{d}x \\ &= \sum_k \int_{-\infty}^{\infty} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)\,\mathrm{d}x \\ &= \sum_k \pi_k \int_{-\infty}^{\infty} \mathcal{N}(x|\mu_k, \Sigma_k)\,\mathrm{d}x \\ &= \sum_k \pi_k \mu_k = \mu \qquad (1.1)\end{aligned}$$

$$\begin{aligned}\mathrm{Cov}[x] &= \mathbb{E}[xx^\intercal] - \mathbb{E}[x]\mathbb{E}[x]^\intercal \\ &= \mathbb{E}[xx^\intercal] - \mu\mu^\intercal && \text{(by 1.1)} \\ &= \mu\mu^\intercal + \sum_k \Sigma_k - \mu\mu^\intercal && \text{(by Bishop 2.62)} \\ &= \sum_k \Sigma_k = \Sigma\end{aligned}$$

Problem 2

Given $\Sigma_k = \sigma^2 I, \sigma^2 \to 0$, we have for the exponential in $\mathcal{N}(\cdot, \cdot)$

$$e^{-\frac{1}{2}(x_i-\mu_k)^\intercal(\sigma^2 I)^{-1}(x_i-\mu_k)} = e^{-\frac{1}{2\sigma^2}(x_i-\mu_k)^\intercal(x_i-\mu_k)} \to 0 \qquad (\sigma^2 \to 0)$$

E-Step

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

It's important to notice that if a datapoint lies far away from a centroid, i.e. $x_i - \mu_k >> 0$ then the exponential term associated to this Gaussian goes much faster to zero than the other ones. Thus for a datapoint x_i we have

$$\gamma(z_{ik}) = \begin{cases} 1, \text{ if } \mathrm{argmin}_k \quad ||x_i - \mu_k||_2 \\ 0, \text{ else} \end{cases} \qquad (2.1)$$

M-Step

Thus the estimates $\mu_k^{\text{new}}, \Sigma_k^{\text{new}}, \pi_k^{\text{new}}$ (cf. slide 29, lecture) become

$$\mu_k^{\text{new}} = \frac{1}{N_k}\sum_i^N z_{ik}x_i$$
$$\Sigma_k^{\text{new}} = \sigma^2 I \qquad \text{(by 2.1 this parameter stays the same as initialized in the beginning)}$$
$$\pi_k^{\text{new}} = \frac{N_k}{N}, N_k = \sum_i^N z_{ik}$$

Thus we see in the case $\Sigma_k = \sigma^2 I, \sigma^2 \to 0$, the EM-Algorithm is equivalent to Lloyd's algorithm.

Problem 3

See below.

11_homework_clustering

January 22, 2018

1 Programming assignment 11: Gaussian Mixture Model

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.mlab as mlab
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline

        from scipy.stats import multivariate_normal
```

1.1 Your task

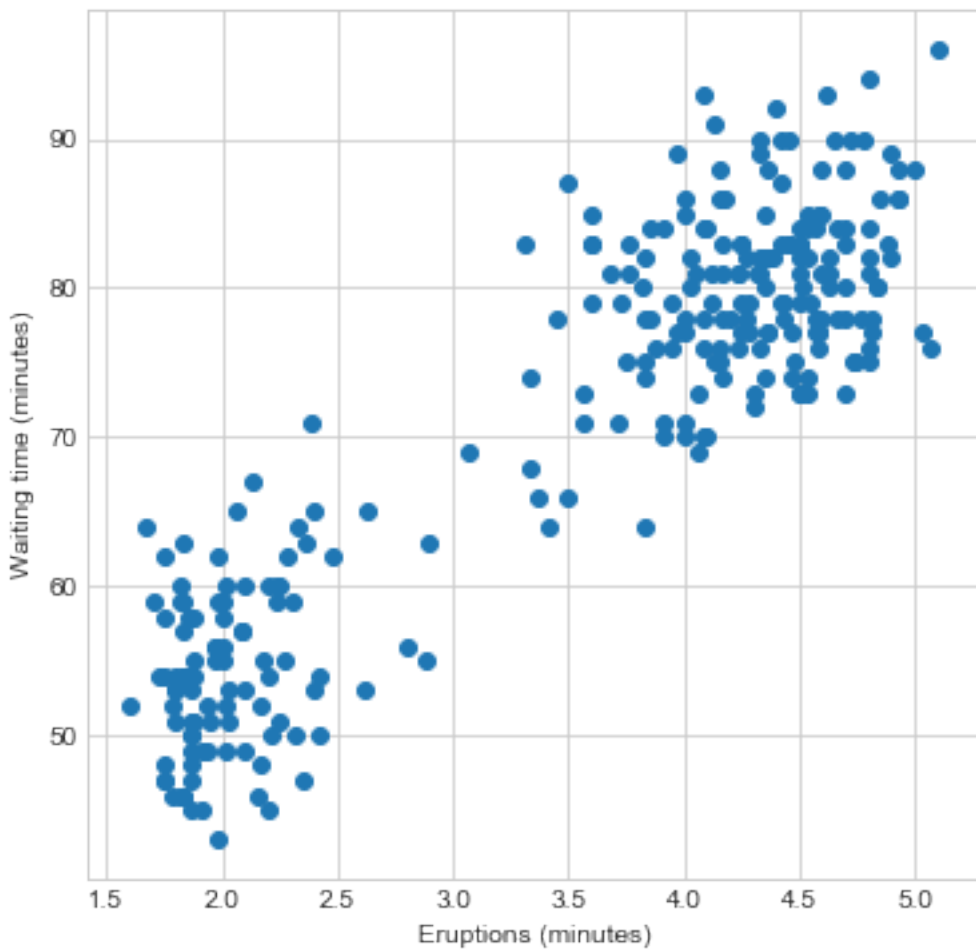
In this homework sheet we will implement Expectation-Maximization algorithm for learning & inference in a Gaussian mixture model.

We will use the dataset containing information about eruptions of a geyser called "Old Faithful". The dataset in suitable format can be downloaded from Piazza.

As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.2 Generate and visualize the data

```
In [3]: X = np.loadtxt('faithful.txt')
        plt.figure(figsize=[6, 6])
        plt.scatter(X[:, 0], X[:, 1])
        plt.xlabel('Eruptions (minutes)')
        plt.ylabel('Waiting time (minutes)')
        plt.show()
```



1.3 Task 1: Normalize the data

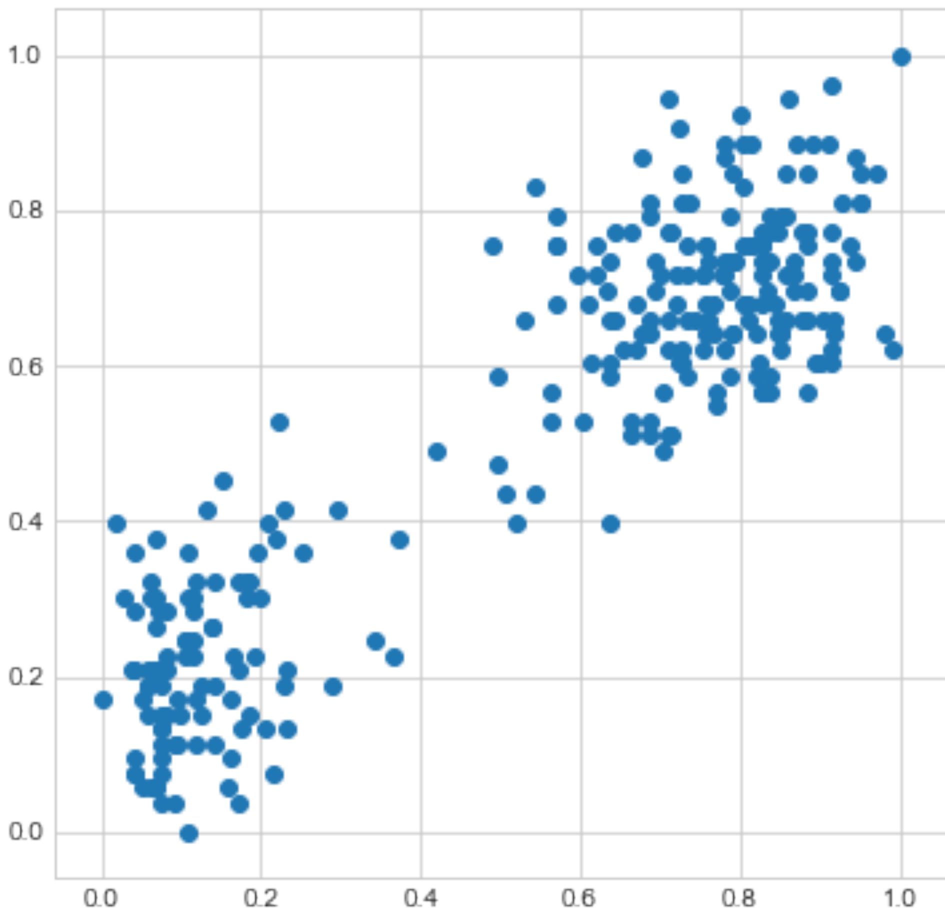
Notice, how the values on two axes are on very different scales. This might cause problems for our clustering algorithm.

Normalize the data, such that it lies in the range $[0, 1]$ along each dimension (each column of X).

```
In [5]: def normalize_data(X):
            """Normalize data such that it lies in range [0, 1] along every dimension.

            Parameters
            ----------
            X : np.array, shape [N, D]
                Data matrix, each row represents a sample.

            Returns
            -------
```

```
    X_norm : np.array, shape [N, D]
        Normalized data matrix.
    """
    X_norm = (X - np.min(X, axis=0)) / \
    (np.max(X, axis=0) - np.min(X, axis=0))
    return X_norm
```

In [6]:
```
plt.figure(figsize=[6, 6])
X_norm = normalize_data(X)
plt.scatter(X_norm[:, 0], X_norm[:, 1]);
```



1.4 Task 2: Compute the log-likelihood of GMM

Here and in some other places, you might want to use the function `multivariate_normal.pdf` from the `scipy.stats` package.

In [13]:
```
def gmm_log_likelihood(X, means, covs, mixing_coefs):
    """Compute the log-likelihood of the data under current parameters setting.
```

```
    Parameters
    ----------
    X : np.array, shape [N, D]
        Data matrix with samples as rows.
    means : np.array, shape [K, D]
        Means of the GMM (\mu in lecture notes).
    covs : np.array, shape [K, D, D]
        Covariance matrices of the GMM (\Sigma in lecuture notes).
    mixing_coefs : np.array, shape [K]
        Mixing proportions of the GMM (\pi in lecture notes).

    Returns
    -------
    log_likelihood : float
        log p(X | \mu, \Sigma, \pi) - Log-likelihood of the data under the given GMM.
    """
    log_likelihood = 0.0
    N, D = X.shape
    K, D = means.shape

    for k in range(K):
        log_likelihood += mixing_coefs[k] * \
        multivariate_normal.pdf(x=X, mean=means[k, :], \
                                cov=covs[k, :, :])

    return np.sum(np.log(log_likelihood))
```

1.5 Task 3: E step

```
In [18]: def e_step(X, means, covs, mixing_coefs):
    """Perform the E step.

    Compute the responsibilities.

    Parameters
    ----------
    X : np.array, shape [N, D]
        Data matrix with samples as rows.
    means : np.array, shape [K, D]
        Means of the GMM (\mu in lecture notes).
    covs : np.array, shape [K, D, D]
        Covariance matrices of the GMM (\Sigma in lecuture notes).
    mixing_coefs : np.array, shape [K]
        Mixing proportions of the GMM (\pi in lecture notes).

    Returns
    -------
```

```
    responsibilities : np.array, shape [N, K]
        Cluster responsibilities for the given data.
    """
    N, D = X.shape
    K, D = means.shape
    responsibilities = np.zeros((N, K))
    normalizer = np.zeros((N,))

    for j in range(K):
        normalizer += mixing_coefs[j] * \
        multivariate_normal.pdf(X, mean=means[j, :], cov=covs[j, :, :])

    for k in range(K):
        responsibilities[:, k] = (mixing_coefs[k] * \
                                  multivariate_normal.pdf(
                                      X, mean=means[k, :],
                                      cov=covs[k, :, :])) / normalizer
    return responsibilities
```

1.6 Task 4: M step

```
In [17]: def m_step(X, responsibilities):
    """Update the parameters \theta of the GMM to maximize E[log p(X, Z | \theta)].

    Parameters
    ----------
    X : np.array, shape [N, D]
        Data matrix with samples as rows.
    responsibilities : np.array, shape [N, K]
        Cluster responsibilities for the given data.

    Returns
    -------
    means : np.array, shape [K, D]
        Means of the GMM (\mu in lecture notes).
    covs : np.array, shape [K, D, D]
        Covariance matrices of the GMM (\Sigma in lecuture notes).
    mixing_coefs : np.array, shape [K]
        Mixing proportions of the GMM (\pi in lecture notes).

    """
    N, D = X.shape
    N, K = responsibilities.shape

    means, covs, mixing_coefs = np.zeros((K, D)), np.zeros((K, D, D)), np.zeros((K,))
    Nk = np.zeros((K, ))

    for k in range(K):
```

```
        Nk[k] = np.sum(responsibilities[:, k])

    for k in range(K):
        means[k, :] = np.dot(responsibilities[:, k][np.newaxis, :], X) / Nk[k]


    for k in range(K):
        for n in range(N):
            covs[k, :, :] += responsibilities[n, k] * \
            np.dot((X[n, :] - means[k, :])[:, np.newaxis],
                   (X[n, :] - means[k, :])[np.newaxis, :])
        covs[k, :, :] = covs[k, :, :] / Nk[k]

    for k in range(K):
        mixing_coefs[k] = Nk[k] / N

    return means, covs, mixing_coefs
```

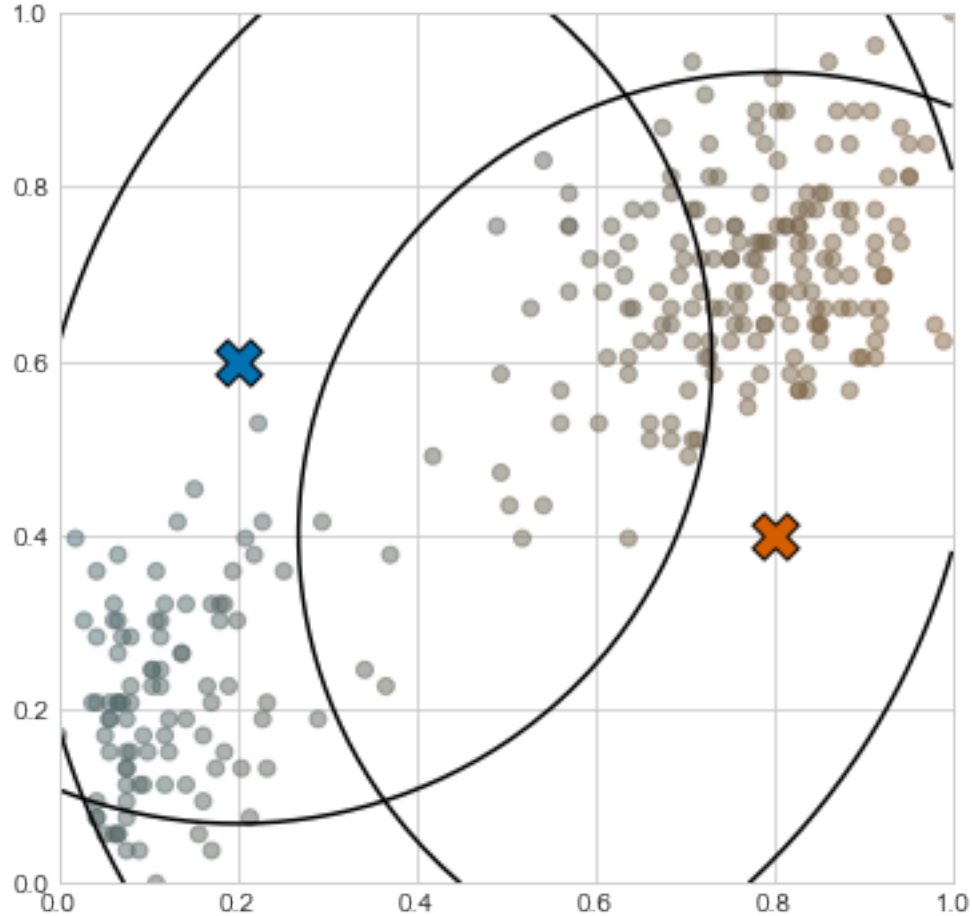
1.7 Visualize the result

```
In [12]: def plot_gmm_2d(X, responsibilities, means, covs, mixing_coefs):
             """Visualize a mixture of 2 bivariate Gaussians.

             This is badly written code. Please don't write code like this.
             """
             plt.figure(figsize=[6, 6])
             palette = np.array(sns.color_palette('colorblind', n_colors=3))[[0, 2]]
             colors = responsibilities.dot(palette)
             # Plot the samples colored according to p(z|x)
             plt.scatter(X[:, 0], X[:, 1], c=colors, alpha=0.5)
             # Plot locations of the means
             for ix, m in enumerate(means):
                 plt.scatter(m[0], m[1], s=300, marker='X', c=palette[ix],
                             edgecolors='k', linewidths=1,)
             # Plot contours of the Gaussian
             x = np.linspace(0, 1, 50)
             y = np.linspace(0, 1, 50)
             xx, yy = np.meshgrid(x, y)
             for k in range(len(mixing_coefs)):
                 zz = mlab.bivariate_normal(xx, yy, np.sqrt(covs[k][0, 0]),
                                            np.sqrt(covs[k][1, 1]),
                                            means[k][0], means[k][1], covs[k][0, 1])
                 plt.contour(xx, yy, zz, 2, colors='k')
             plt.xlim(0, 1)
             plt.ylim(0, 1)
             plt.show()
```

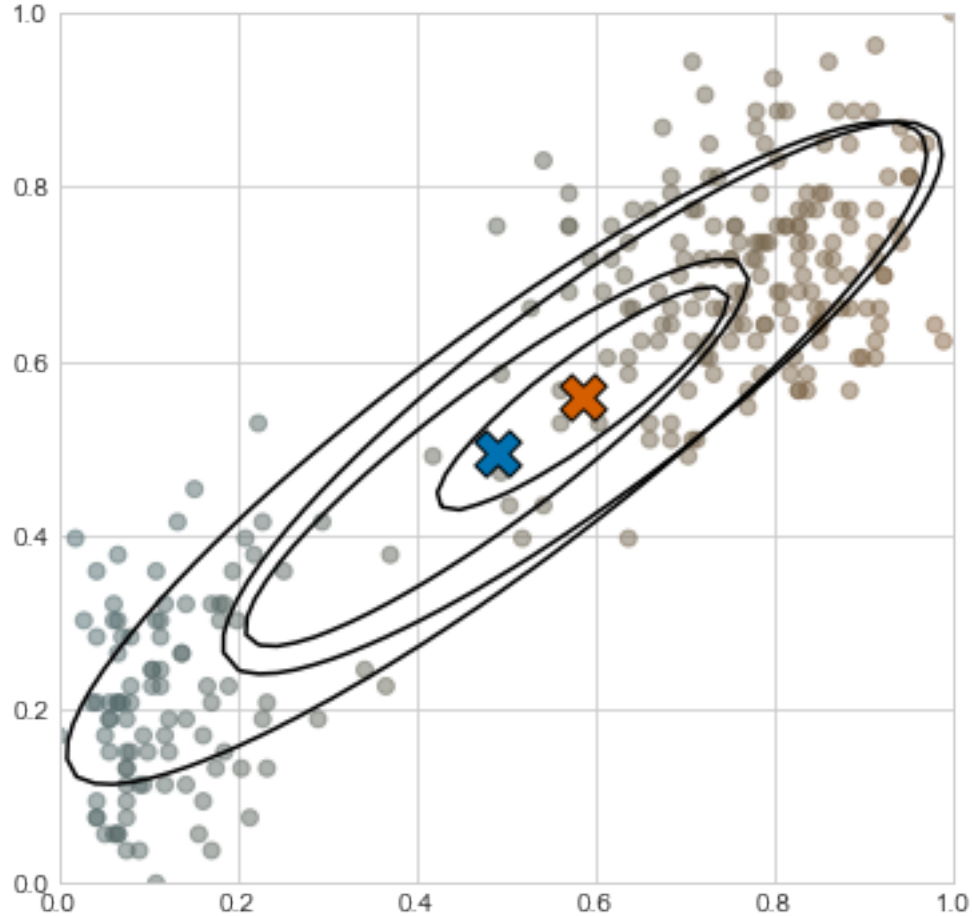
1.8 Run the EM algorithm

In [119]:
```
X_norm = normalize_data(X)
max_iters = 20

# Initialize the parameters
means = np.array([[0.2, 0.6], [0.8, 0.4]])
covs = np.array([0.5 * np.eye(2), 0.5 * np.eye(2)])
mixing_coefs = np.array([0.5, 0.5])

old_log_likelihood = gmm_log_likelihood(X_norm, means, covs, mixing_coefs)
responsibilities = e_step(X_norm, means, covs, mixing_coefs)
print('At initialization: log-likelihood = {0}'
      .format(old_log_likelihood))
plot_gmm_2d(X_norm, responsibilities, means, covs, mixing_coefs)

# Perform the EM iteration
for i in range(max_iters):
    responsibilities = e_step(X_norm, means, covs, mixing_coefs)
    means, covs, mixing_coefs = m_step(X_norm, responsibilities)
    new_log_likelihood = gmm_log_likelihood(X_norm, means, covs, mixing_coefs)
    # Report & visualize the optimization progress
    print('Iteration {0}: log-likelihood = {1:.2f}, improvement = {2:.2f}'
          .format(i, new_log_likelihood, new_log_likelihood - old_log_likelihood))
    old_log_likelihood = new_log_likelihood
    plot_gmm_2d(X_norm, responsibilities, means, covs, mixing_coefs)
```

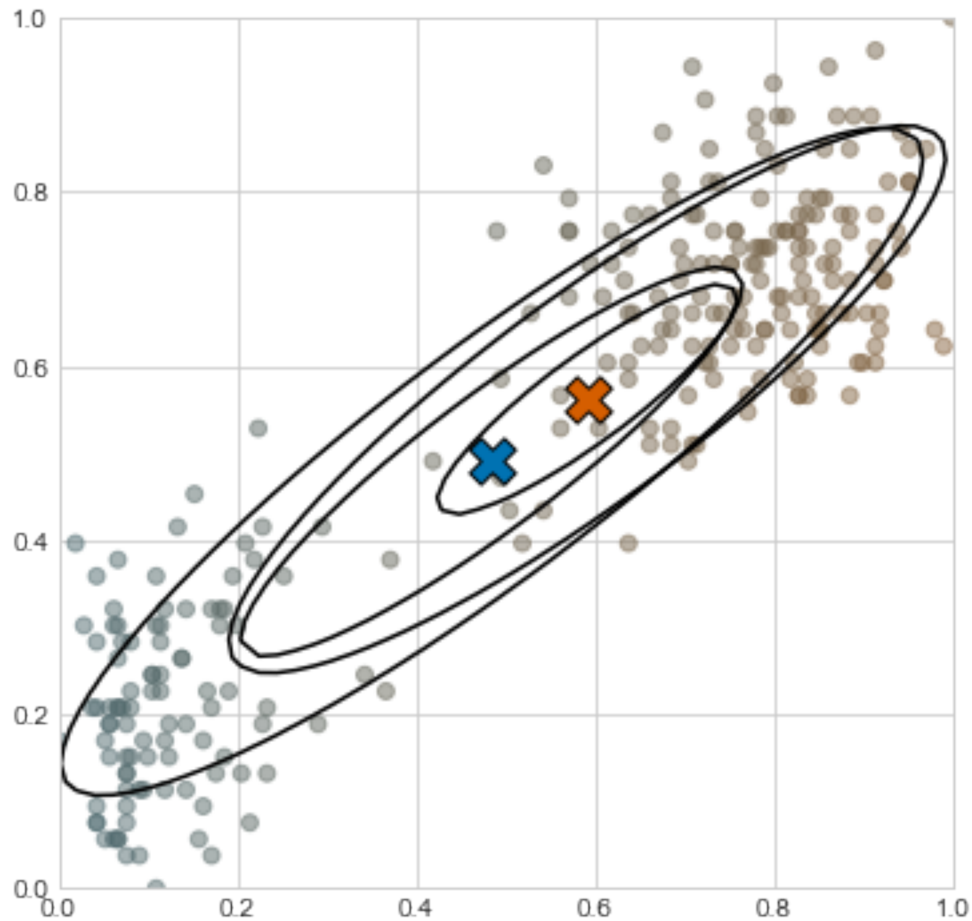
```
At initialization: log-likelihood = -382.7055152420654
```



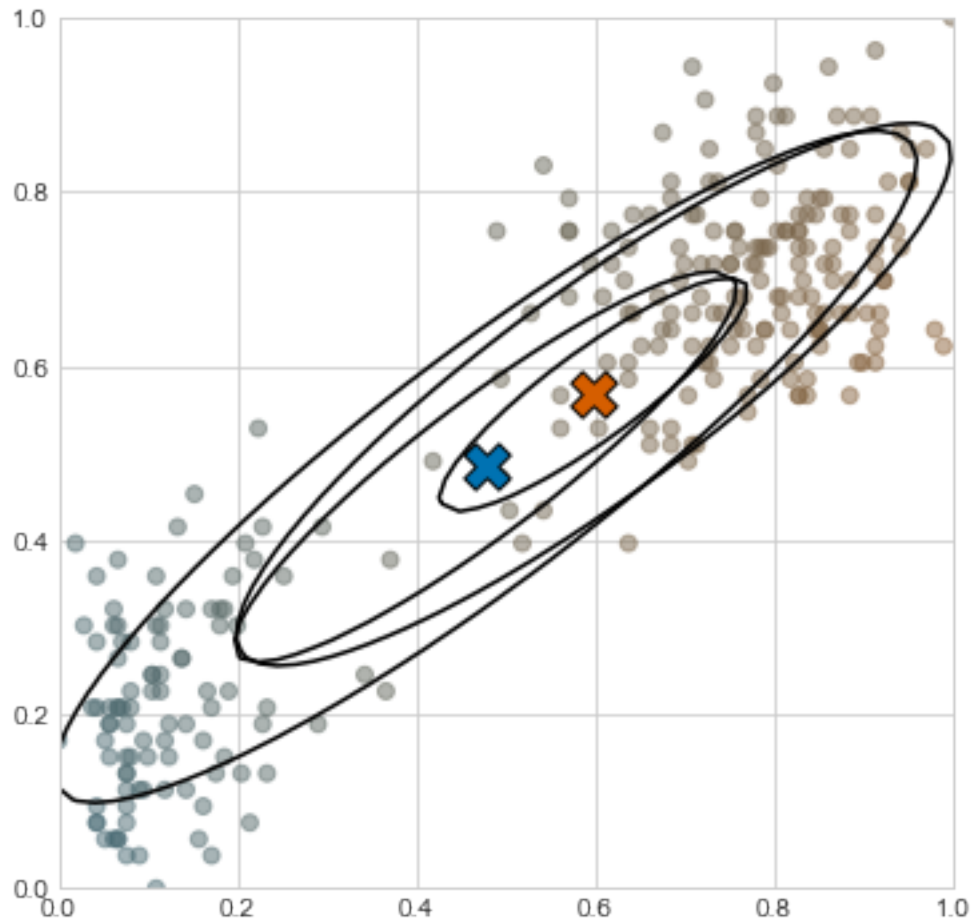

```
Iteration 0: log-likelihood = 131.29, improvement = 513.99
```



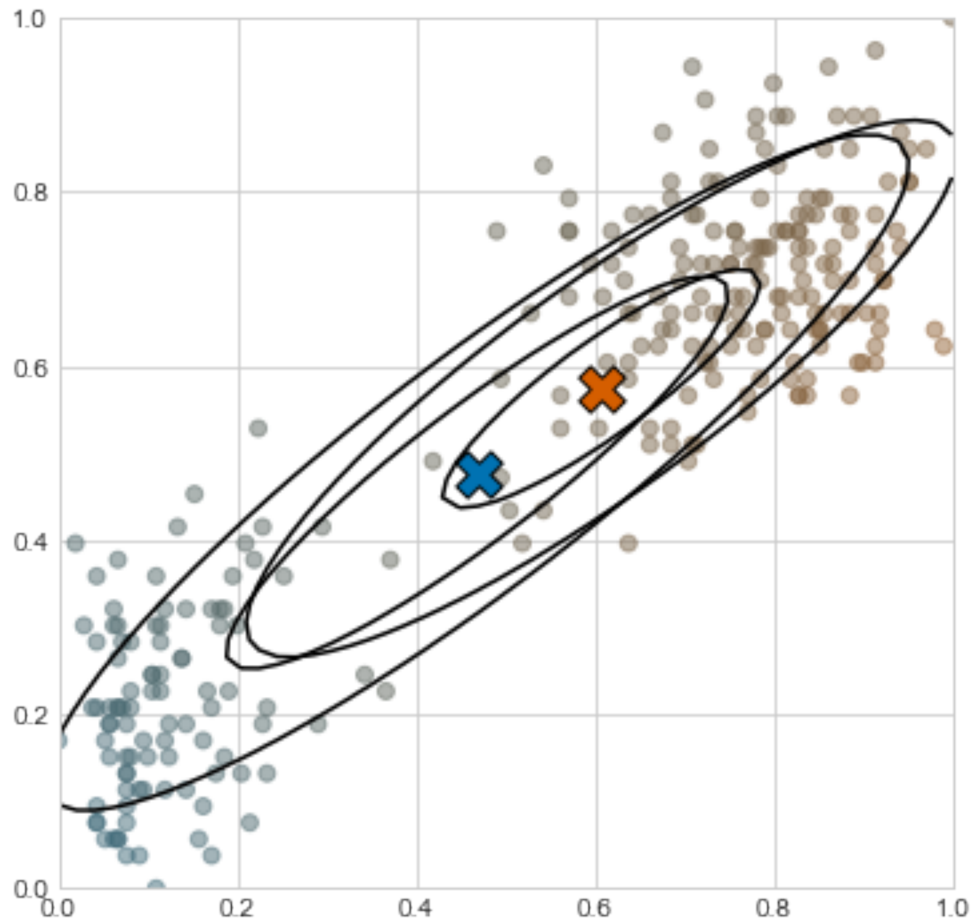

```
Iteration 1: log-likelihood = 131.48, improvement = 0.19
```



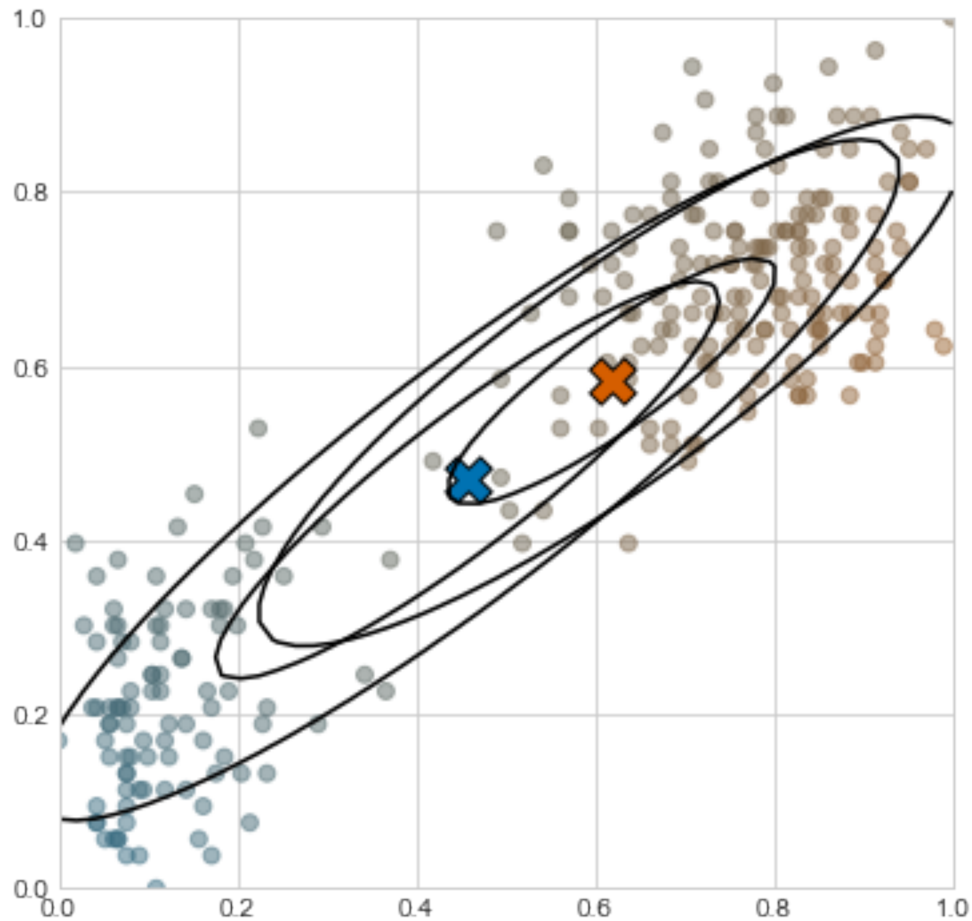

```
Iteration 2: log-likelihood = 131.75, improvement = 0.27
```



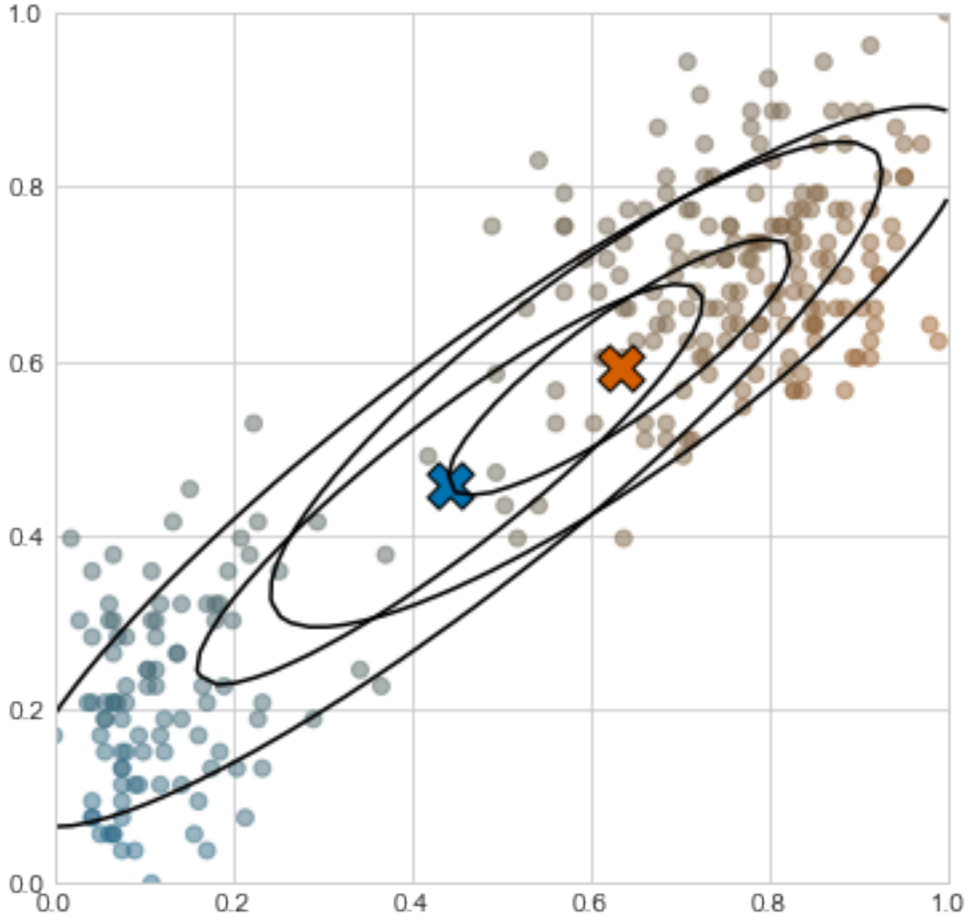
Iteration 3: log-likelihood = 132.15, improvement = 0.40



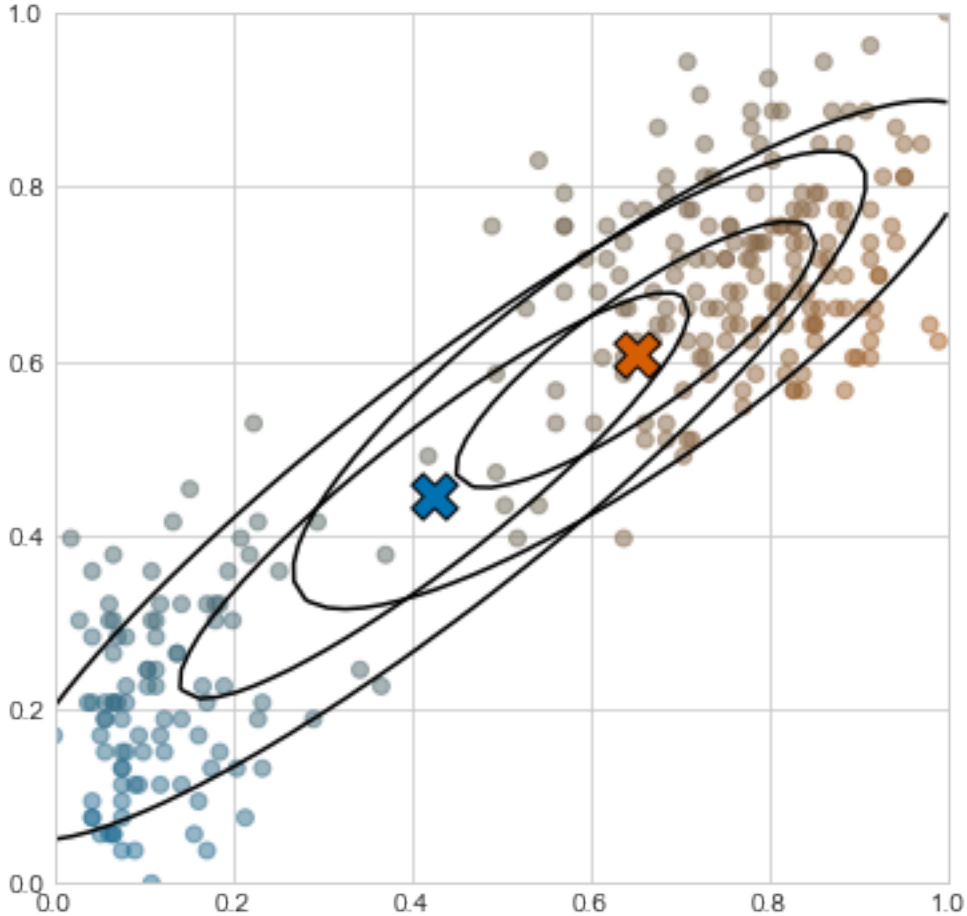

```
Iteration 4: log-likelihood = 132.77, improvement = 0.62
```



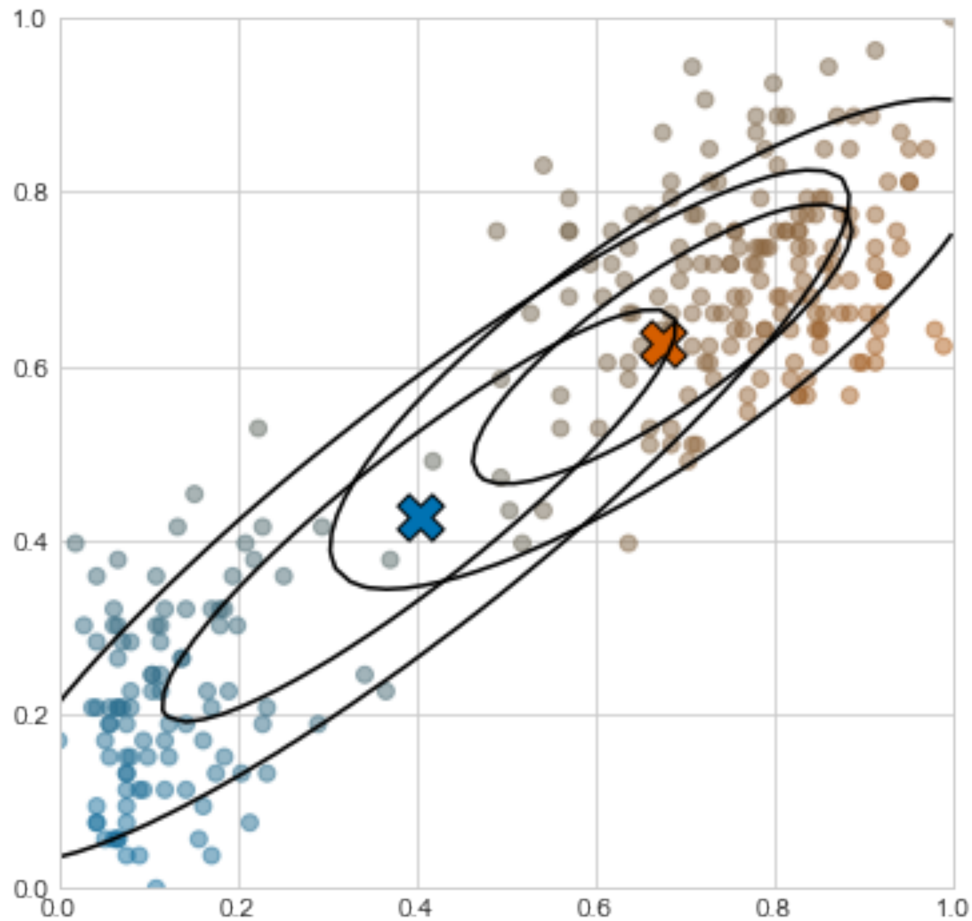

```
Iteration 5: log-likelihood = 133.81, improvement = 1.04
```



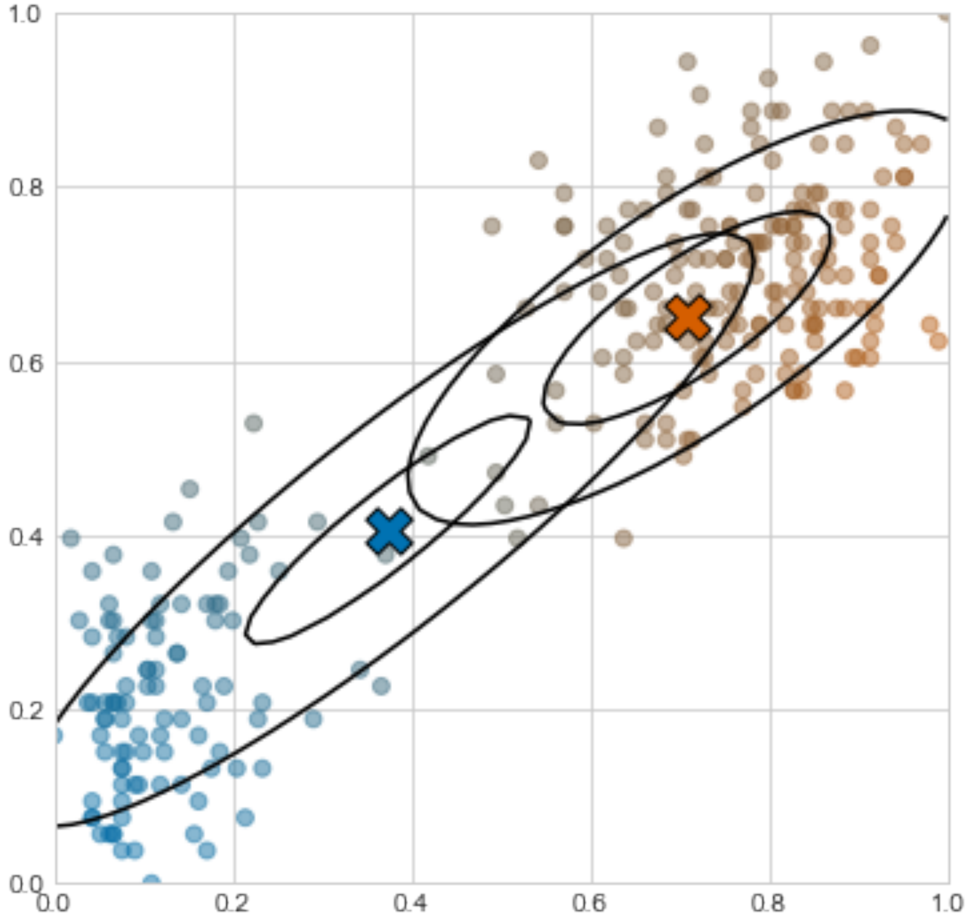

```
Iteration 6: log-likelihood = 135.74, improvement = 1.93
```



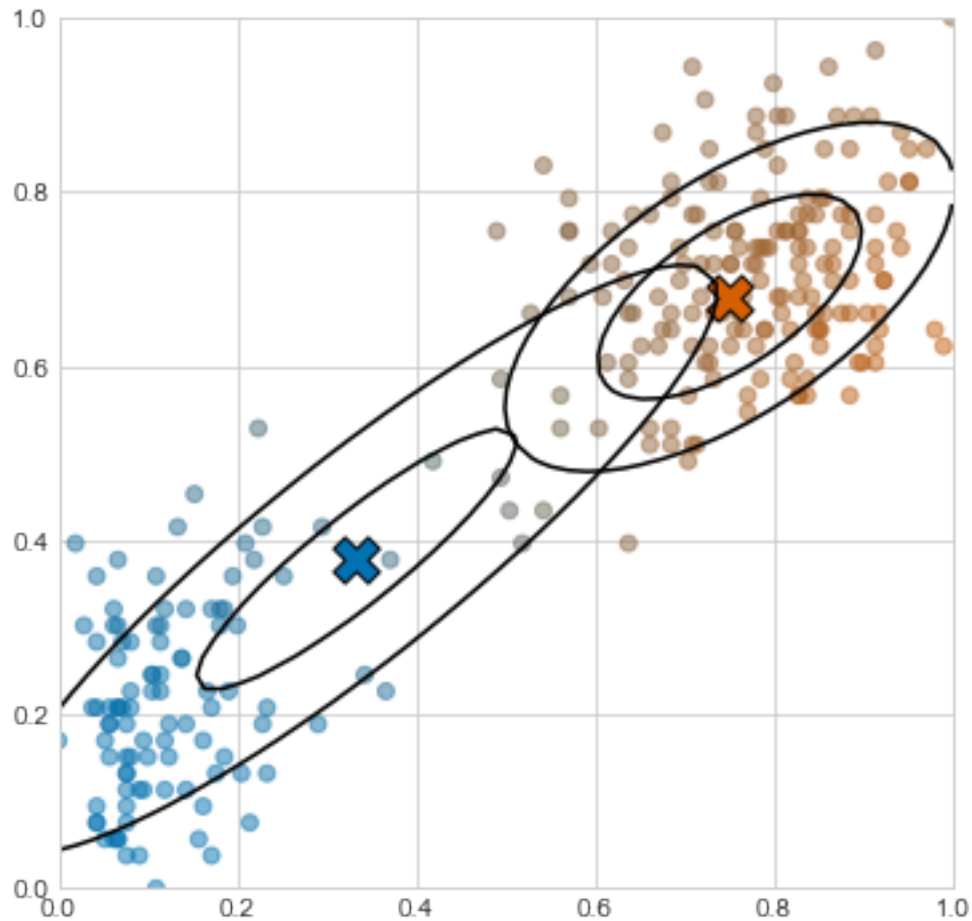

```
Iteration 7: log-likelihood = 139.88, improvement = 4.14
```



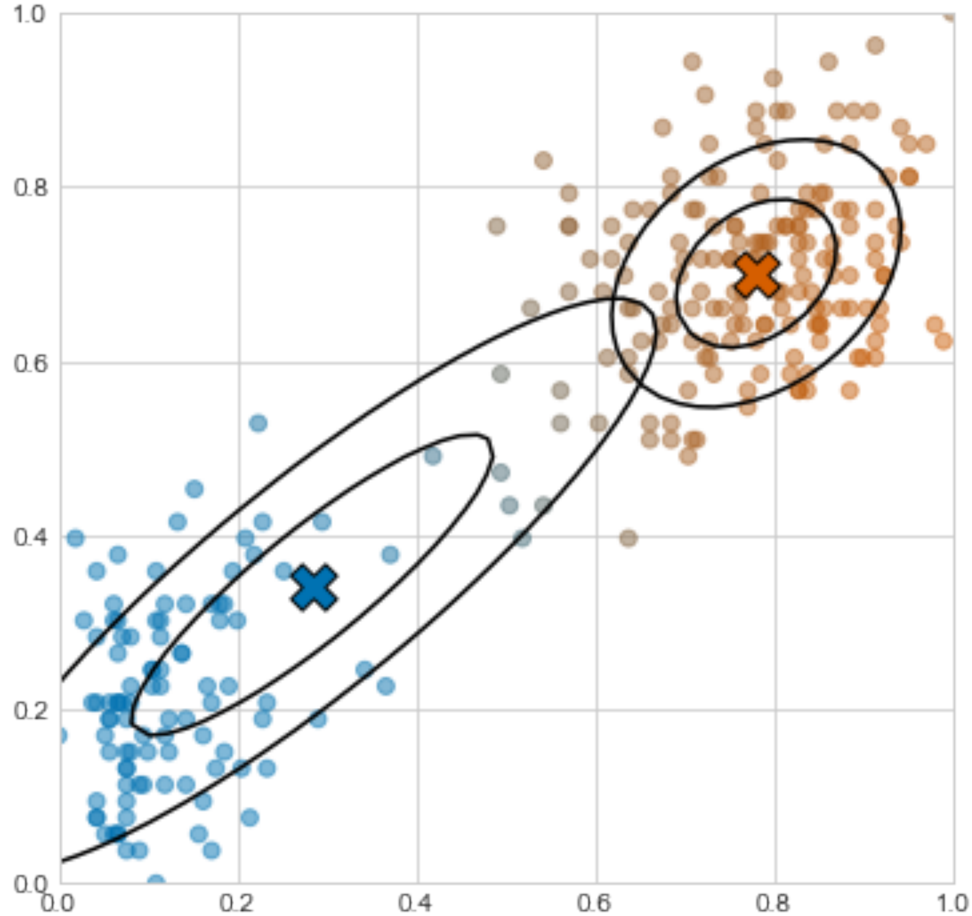

```
Iteration 8: log-likelihood = 150.67, improvement = 10.79
```



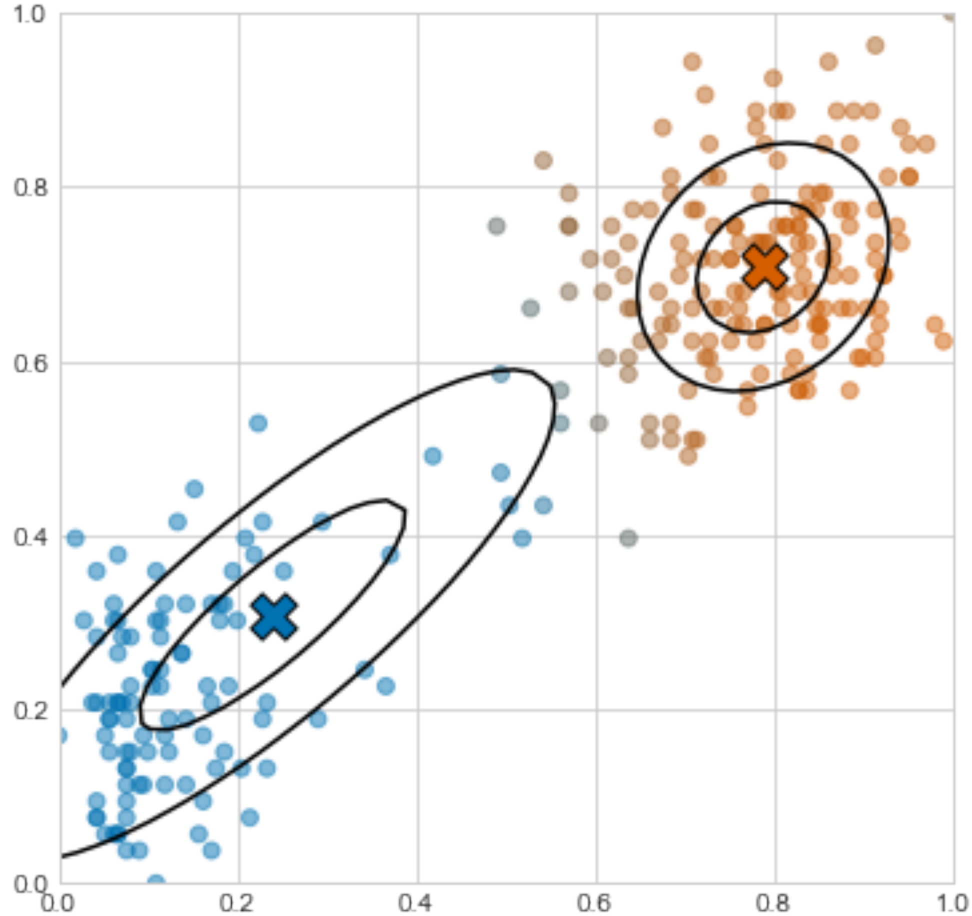

```
Iteration 9: log-likelihood = 181.12, improvement = 30.45
```



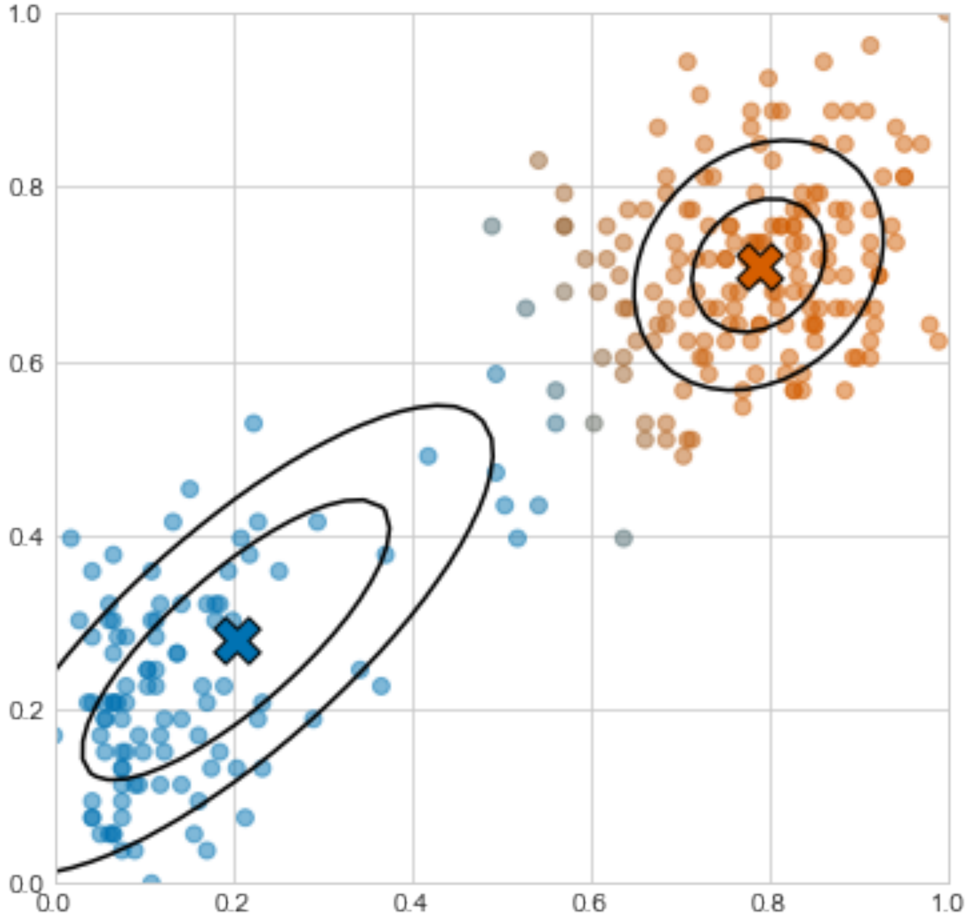

```
Iteration 10: log-likelihood = 220.93, improvement = 39.81
```



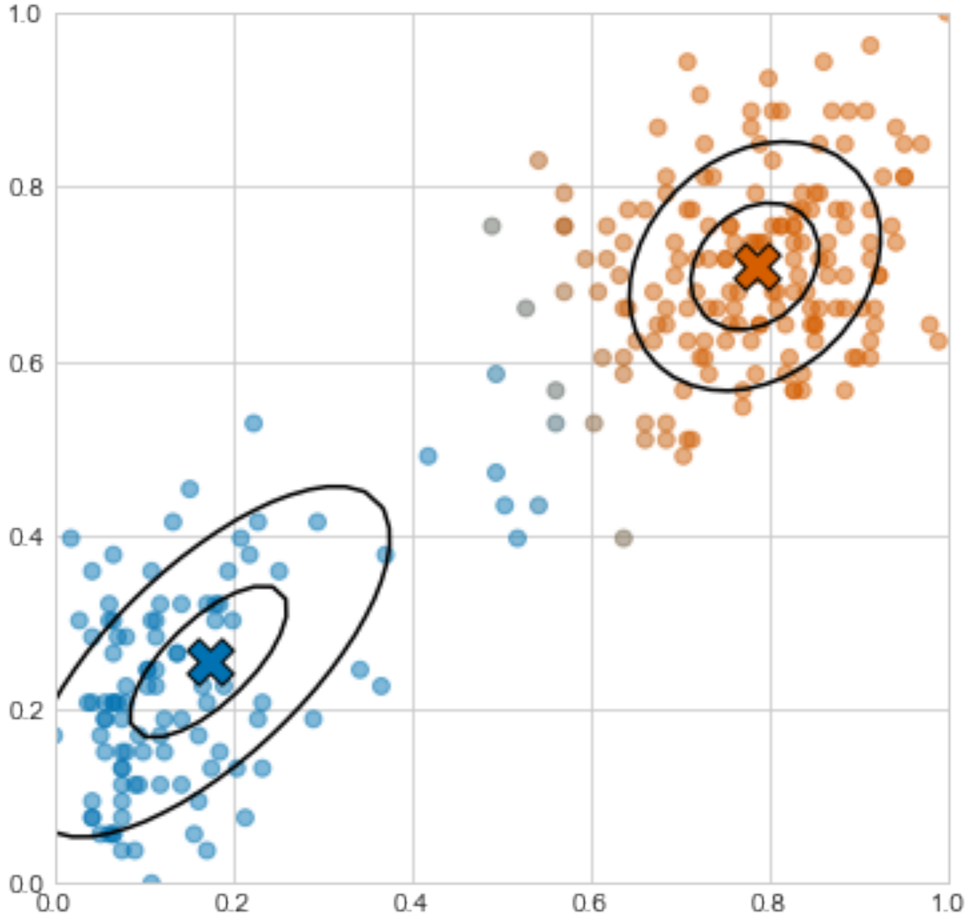

```
Iteration 11: log-likelihood = 234.06, improvement = 13.14
```



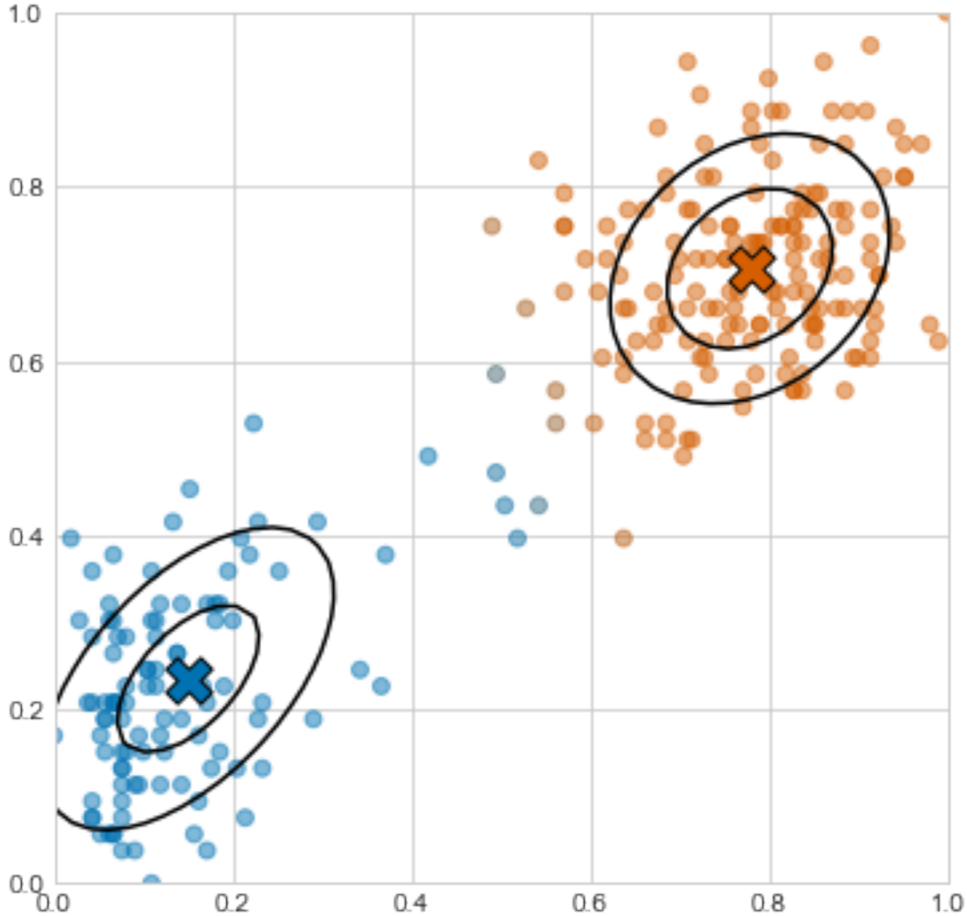

```
Iteration 12: log-likelihood = 244.83, improvement = 10.77
```



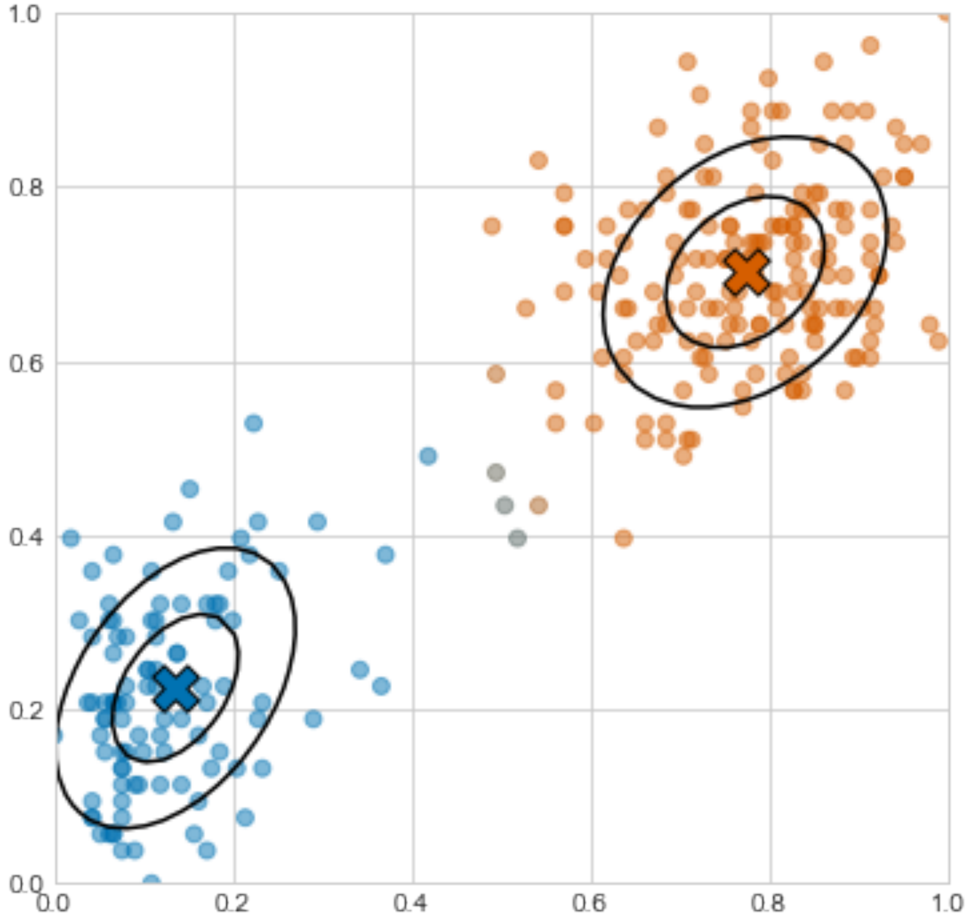

```
Iteration 13: log-likelihood = 258.67, improvement = 13.84
```



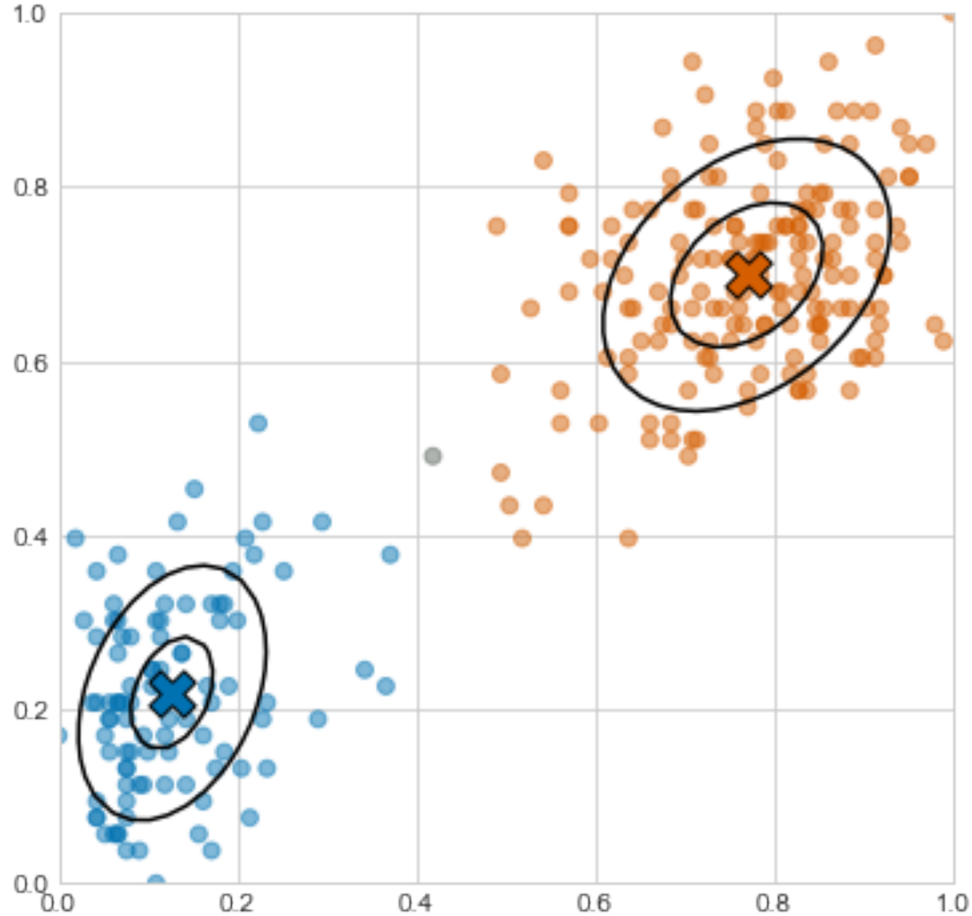

```
Iteration 14: log-likelihood = 272.91, improvement = 14.23
```



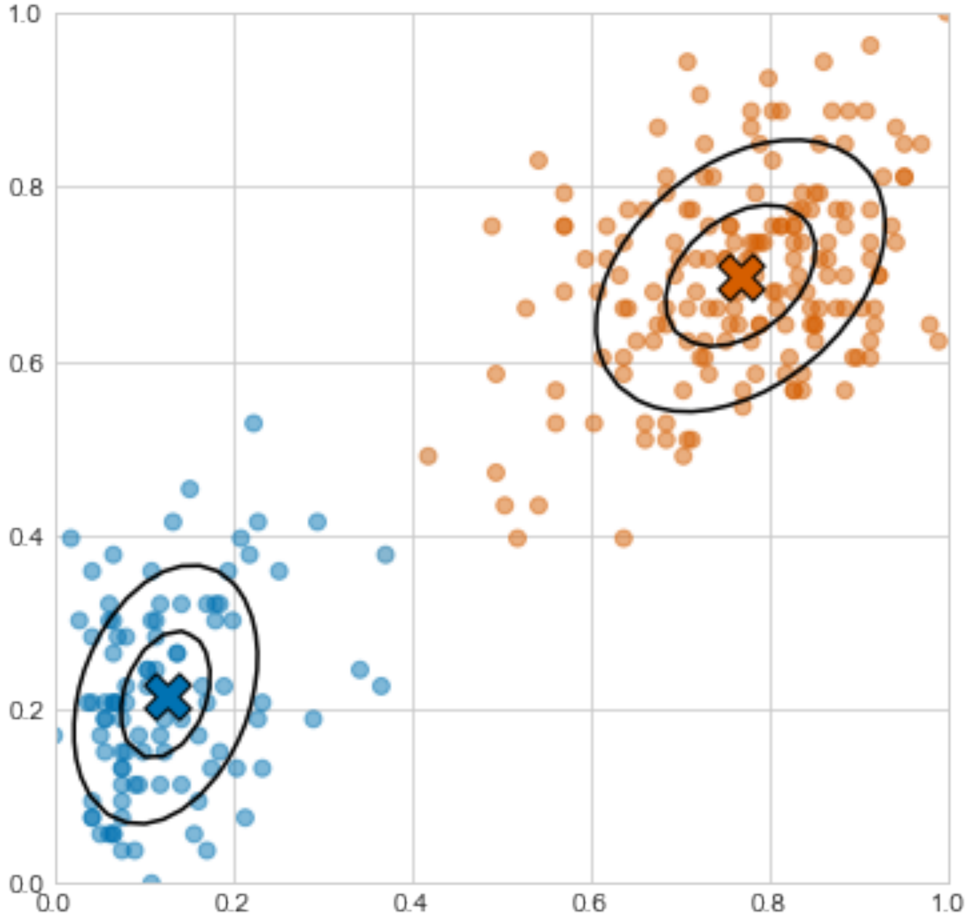

```
Iteration 15: log-likelihood = 284.29, improvement = 11.38
```



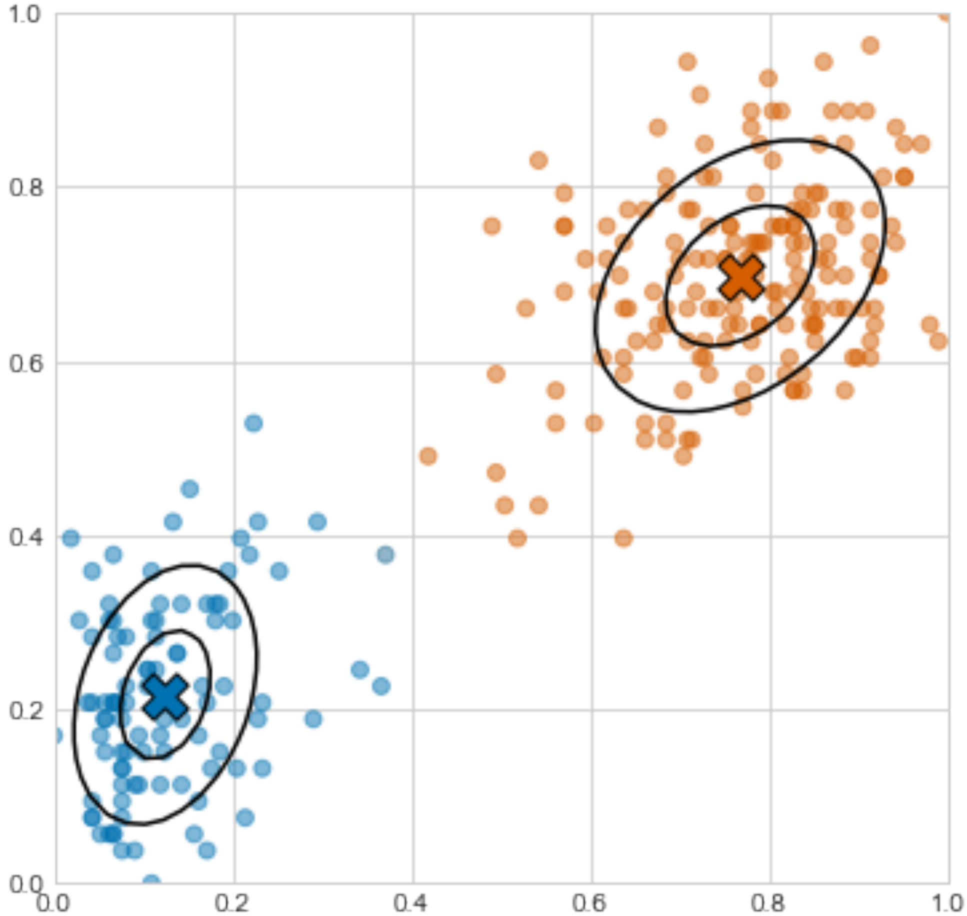

```
Iteration 16: log-likelihood = 289.94, improvement = 5.65
```



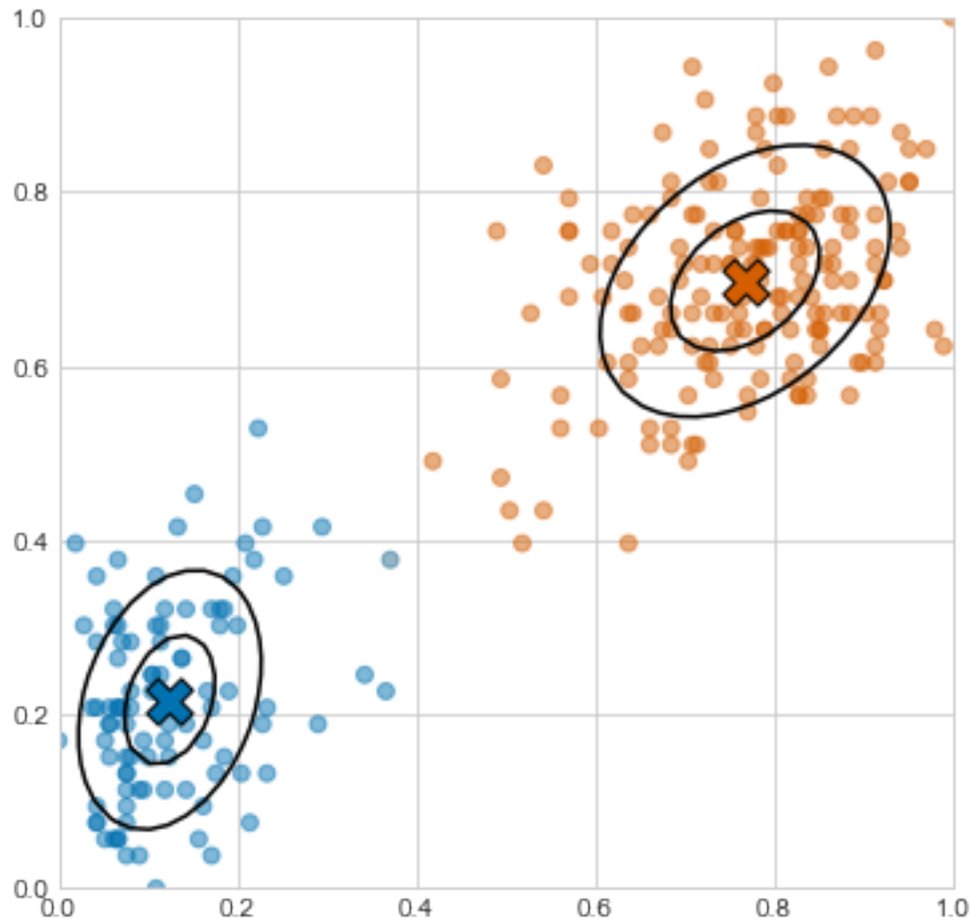

```
Iteration 17: log-likelihood = 290.39, improvement = 0.45
```




```
Iteration 18: log-likelihood = 290.41, improvement = 0.01
```




```
Iteration 19: log-likelihood = 290.41, improvement = 0.00
```



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0